Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Vojtěch Horký

Webová aplikace pro komunikaci mezi učiteli a studenty

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Rudolf Kryl
Studijní program: Informatika, Programování

2009

Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS

Vojtěch Horký

Web Communication between Teachers and Students

Department of Software and Computer Science Education

Supervisor: RNDr. Rudolf Kryl

Study Program: Computer Science, Programming

2009

I would like to thank my supervisor, Rudolf Kryl, for the countless advices and the time spent on guiding me on this thesis.

I hereby declare that I wrote the thesis myself using only the referenced sources. I agree with lending the thesis.

Prague, May 28, 2009 Vojtěch Horký

Contents

Název práce: Webová aplikace pro komunikaci mezi učiteli a studenty
Autor: Vojtěcy Horký
Katedra (ústav): Matematicko-fyzikální fakulta univerzity Karlovy
Vedoucí bakalářské práce: RNDr. Rudolf Kryl, Kabinet software a výuky informatiky
e-mail vedoucího: Rudolf.Kryl@mff.cuni.cz

Abstrakt: V rámci práce bude vytvořen softwarový nástoj Lakmus pro usnadnění komunikace mezi učiteli a jejich žáky i mezi žáky navzájem. Nástroj bude vytvořen tak, aby byl snadno konfigurovatelný a rozšiřitelný o nové funce.

Klíčová slova: komunikace PHP SQL

Title: Web Communication between Teachers and Students
Author: Vojtěch Horký
Department: Faculty of Mathematics and Physics, Charles University
Supervisor: RNDr. Rudolf Kryl, Department of Software and Computer Science Education
Supervisor's e-mail address: Rudolf.Kryl@mff.cuni.cz

Abstract: In the framework of the bachelor thesis will be created a tool (named Lakmus) to facilitate communication between teachers and students as well as among students. The tool will be designed as easily configurable and extensible.

Keywords: communcation PHP SQL

Chapter 1

Motivation

The initial motivation to create Lakmus was frustration (sometimes mixed with a disgust). During the first two years at University I attended to more than 15 lectures that have some information connected to them available on the Internet. The stumbling-block was that these information were spread on homepages of the lecturers and sometimes were very difficult to reach without knowing the exact web address. Another common thing among these pages was that they share much of their functionality – there were some kind of score tables with our results on examinations in one third of them, many had some kind of tool for file upload (with our homework etc).

Although these pages worked flawlessly and very perfectly suitable for the job alone, when thinking on more general level, lot of the functionality was duplicated or hard to reach.

So, when deciding what to use for my "Year project" it dawned on me that a web tool that would cover in these tools might be a good idea.

There were two basic requirements. The application has to be simple on its usage because otherwise no-one would be ecstatic about starting using it when a working solution already existed. And it has to allow users easy extending, thus getting nearer their needs. Among other requirements is that it shall be accessible even to handicapped users and shall be browser independent.

When started programming it, a problem arouse – the user might be eventually submerged with data (of any kind) and would abandon the tool. To prevent such situation, the concept of labels was introduced.

1.1 Labels

Labels – in some applications called tags – are a virtual representation of yellow sticky notes. You can post them on almost anything and you can filter displayed data with them.

In Lakmus, labels are private and – as mentioned above – can be stuck to almost anything (in any case, to anything where it makes sense to have it labeled). There is no limit on the number of labels that each user has created and users do not need to manage the labels explicitly. That means that when user wants to label some object, he can either select from a list of existing labels or simply enter the new label title and label is attached (and if necessary – though this process is hidden – created).

There is one disadvantage though – with the amount of labels unlimited, only labels that are really attached are displayed in filtering bars[1]. That alone makes sense but sometimes it may be a problem – consider a situation where you use labels to evaluate quality of some homework. If no user submits the "Perfect" solution, this label will not appear in the label listing which might be confusing. Unfortunately, there is no easy way to change this behavior (displaying all labels is nonsense as it would submerge user with too much data; letting user manually select which labels to display is impractical any any "smart" filtering[2] ended in displaying too much labels[3]). Finally, I decided to leave it that way as it seems to me as the best option available.

The filtering itself is in most cases activated with a "standardized" control that supports multi-label filtering. That means that selecting more labels will filter already displayed objects. I decided for this "and" behavior (with the "or" one unavailable) because usually you want to narrow down displayed data rather than widen your search result.

[1] Filtering bar is only a list of labels that are attached to the category of objects that are currently listed. By clicking on the label, the list of objects is narrowed only to those having selected label attached.

[2] E.g. display labels that are used the most.

[3] At least those that came to my mind.

Chapter 2

Why I have choose web application and why PHP was the good choice

The reason for choosing a web application is because of its portability. All you need to access it is a web browser. Although there are many different browsers and sometimes debugging a layout is a tedious task, it is much simpler than debugging normal (i.e. client side) application, especially when we take in account that students use different operating systems and architectures and are very reluctant to use any software that is not native. Luckily, even the oddest architecture usually have a web browser.

2.1 Choosing PHP

One of the main goals when creating Lakmus was to motivate users write their own applications, thus allowing them to extend Lakmus to their needs. Because of this, one of the aspects of the selected programming language was its simplicity and popularity[4]. PHP suited both of these criteria.

However, there are several aspects that are against its usage. First, it is an interpreted language (more or less) which may effect performance. Secondly, it is not very strict language in terms of type checking which could lead to an unreadable code sometimes (see chapter 3 to see how this problem is dealt with).

2.2 Using frameworks vs. writing from scratch

The initial motive from writing Lakmus from scratch was simple unwillingness to learn some of the PHP frameworks. However – to defend myself – it was usually because of lack of documentation. I do not want to blame authors of the framework but although I have no problem finding some tutorials on building a simple application (such as a personal blog), tutorials for some advanced features were missing and were replaced by API documentation. Because of this, I was very reluctant to start writing Lakmus using one of these tools, thus committing myself to use it for whole application before knowing that it would be usable even for the complex parts of it.

Secondly, such frameworks causes a performance degradation – but that is not the problem: the problem is that on the Internet there is a lot of benchmarking done but there is no problem finding contradictory results [2] vs. [3].

What I wanted to point out is that when deciding which (or whether) framework to use, I could rely only on some kind of intuition how much I like the code. And after realising that the main goal of some of the frameworks is just to provide a library for simplifying repetitive tasks (such as form creation and validating), I dismissed the idea of the framework-based application idea and decided to write the application "from scratch". This decision was also supported by the fact that I already had a tiny library for simplifying work with forms[1] that I was familiar with and I could start using it immediately (though I have to admit that I had to subsequently rewrite this library completely).

There is another point of view that rejects usage of a framework. As stated before, goal of Lakmus was to allow extending it – that means that the person who wants to do such thing first needs to learn a bit of the internals of the application which alone takes some time and effort. And now add to it that this potential developer would have to learn also the principles behind used framework. I am afraid that this approach might be disheartening.

Another argument against frameworks is their complexity. They are designed to be general-purposed and even when using only a small part of them, they load a huge amount of helper classes as a back-end.

[1] I created Phenolphthalein earlier when learning PHP

2.3 External libraries

Although I rejected usage of existing PHP framework, I chose existing libraries for several tasks. Here I would like to give my reasons.

I decided for JQuery for client-side scripting as it is open-source (GPL licenced) and it is a very stable framework that offers wide range of functions. I also tried MooTools and dLite but JQuery won because of good documentation and because of its UI library.

For e-mailing I used the PHPMailer library with little changes. Actually, this library seems to me as very poorly written but I was not able to find any better with the functionality I needed. However, wrapping its classes inside `Lakmus Mailer` makes any further changes quite easy.

I had problem finding a decent library for exports to Excel. Either the library was too simple (e.g. the IAM-XLS) or too big (e.g. the PHP Excel by Microsoft itself). Finally, I ended up with Spreadsheet_WriteExcel that somehow fitted my needs. The only feature I miss is its inability to deal with cross-worksheet references in cell formulas (which is the reason why the scoreboard export look the way it looks).

2.4 Comparison to existing software

When developing a new application, there is always a risk that there already exists a similar application. However, to the best of my knowledge, there is not any similar application that would offer complete access to its internals having the same objective as Lakmus.

Actually, the closest hit is current Student Information System (usually referred to as SIS) with its "Grupíček"[2]. Although it provides several features also available in Lakmus, it is scarcely used by lecturers. Most of the complaints are that it is way to difficult to control it – result being that only fraction of its potential is used. This might be because all the functionality is aggregated in one application that do not provide means of any separation. Bearing that in mind, each application in Lakmus is separated as much as possible from others and neither depend on other ones.

Other tool with similar purpose is CodEx that is used for evaluation of homework tasks of programming lectures. Although CodEx may be "abused" to provide similar functionality as Lakmus, it is targeted at automatic evaluation while

[2]Study group roster

Lakmus expects manual evaluation and thus they shall not be taken as competitors but more as counterparts (omitting the fact that CodEx is regularly used and proved to be useful).

Chapter 3

Highlights of Lakmus

Here I would like to point out things that make Lakmus different from other tools of similar purpose.

3.1 Clean source code & error handling

I do not want to emphasise this advantage too much because neat source code shall be something obvious – not a bonus. However, because PHP is one of the languages where most of the coders just hang together the code and hack it until it works, I would like to point out that in Lakmus I took a more defensive approach to prevent that kind of programming.

PHP is not a statically typed language and it is mostly the possibility to use same variable for different types that obfuscates the code. In addition to this, implicit conversions can cause you a serious trouble – especially with the introduction of the `===` operator. To minimize these problems, I put an assertion-like type checking to the beginning of every public method. Although it does not prevent doing bad things completely, it is a pretty effective counter measure.

Also, each method always returns value of the same type (that is listed in the documentation). There is one exception to that: when informing that some action failed. For example, when asked for details of group with certain id, the method would return structure[1] if the provided id is valid (i.e. group with such id exists) and would return `FALSE` when the id is invalid. I decided to use this approach over exception throwing and leave exceptions only for serious problems and errors

[1] Of course, under PHP terms, it is just an array but I have found a good practice to call it structure – or struct in method signatures – because it tells more about the returned object. After all, we are not returning data about more objects, only structured info about single one.

(mostly on the top level). That means that exception would be rather thrown after receiving this `FALSE` or in situation when there is no mean of recovery (such as invalid URL or database connection problem).

Although I understand that throwing an exception always would be a valid and probably even a cleaner solution (from the language point of view), I decided against it. First, programmer shall check what the function does return and not rely on some catch block seven blocks above. Secondly, when such thing happens it is a good practice to recover as gracefully as possible – in a web application that means, for example, create all the local navigation and links that are not broken by this invalid input – and that would usually mean only catching the exception and then its rethrow with additional HTML formatting[2].

Other guidelines I followed – such as naming conventions etc. – are described in detail in User's manual.

3.2 Layered structure

Lakmus uses three-layer structure when accessing data of (almost) any kind.

First, there is the top one that also takes care of their displaying. On this level, you ask for information with high-level methods such as `getGroupDetails` and you get back a structure with group details. You do not need to worry about what kind of data storage is used and there would be no need for any changes when the database back-end would change.

Then, there is the middle layer that provides access to all data in Lakmus. However, this layer does not call SQL database API directly but rather uses the lowest layer, accessing it again with high-level methods such as `sqlDelete`. Thus, this layer would survive without any modifications a change of the database back-end (SQL engine).

The lowest level is a more or less intelligent wrapper around SQL database API. Current version of Lakmus uses MySQL, however, if needed (e.g. because of performance) a shift to a different one (e.g. PostgreSQL) would mean only rewriting code at this layer. As stated before, apart from mere hiding differences between various databases, the wrapper also provides its own functionality. For example, on SELECT queries it returns whole array thus saving developers from calls to "fetch-row"-like calls in every loop iteration. Although it is a good practice not to fetch all data (from SQL query) at once (due to excessive memory

[2]As a matter of fact, I tried that approach but abandoned it because the code was bloated more than with simple "if" check.

usage) and use the "fetch loop" instead, I decided against it because it would make the middle layer more complicated. Moreover, the result set is usually very small (commonly several kilobytes) and thus the memory is not an issue.

Furthermore, it allows to INSERT and UPDATE data by simply passing a structure (associative array) where index is column name and value is the column value – method then "unpacks" this and creates an SQL statement. However, such functionality is not provided for SELECT queries because they are usually complex and adding another abstraction would only darken their purpose.

I decided to use three-level structure because it is yet simple to understand what happens at each level and also it gives enough means for different levels of abstraction. I found out that although four layers could make sense in some situations there are just too much of them. It led to a situation where most of the time I wondered whether some method shall be in layer 2 or 3 instead of focusing on the code. On the other hand, narrowing layers to only two of them (effectively, having the display layer and a data-access one) would make any data back-end change very time-consuming because everything would have to be rewritten even when the core is the same.

With two layers of data access, it is possible to change SQL engine without affecting the upper layer and even when a more sophisticated change (such as a transformation to a XML-database) would take place there may not be a need for any major rewriting of the code in the upper layer. Of course – having the transfer to XML-database in mind – all the SELECT queries would have to be rewritten to XPath expressions but, for example the INSERT statements could remain almost intact.

3.3 Accessibility

When writing Lakmus, I always bore in mind that no-one shall be denied access to it because of the software they use or because of the way they use the computer.

As a result, generated pages of Lakmus are pretty terse and misses anything that could be called "eye-candy". Each set of links is preceded by a heading informing where the links point to, most links have titles with more detailed description of the action they would invoke.

Also, the page is structured in such way that it could be read from top to bottom without need to return in reading (and, of course, there is a link for skipping the navigation).

For normal users[3], however, the Lakmus provides an attractive interface. Although there is principally nothing extra in such approach, there are several things to point out.

One of them is that the HTML code does not (with exception of the logo) include any decorative images [6]. That has two positive effects – the code is shorter (thus quicker to load) and reading in text browser is easier. On the other hand, users usually want an attractive interface. That is accomplished with per-CSS inserted background images.

Another one are floating helps in forms. In a modern browser, on mouse hovering above the input field, a floating help would appear. On many sites this is done by JavaScript and often the text of the floating help is not places next to the field it belongs to. Effectively, that makes it unusable – and often useless as well – with text browsers or screen readers. For Lakmus, I used different approach: the text of the help (or hint) is placed behind the normal label. The text is enclosed in `small` tag thus indicating it is an extra information and so would be rendered in text-browsers. However, in modern browser, the text would be made hidden and would appear as a floating box only on mouse hovering. Except the fact that it uses the `:hover` class that is not supported for all elements in Microsoft Internet Explorer I would consider it as an optimal solution. The problem with MSIE was eliminated by using the "Whatever:hover" script[7].

3.4 Look-and-feel

When programming in PHP, the developer is often tempted to mix-up the logic of the application with its look by inserting the HTML markup here-and-there in the code. This is a great advantage when prototyping or when writing a smaller project. On the other hand, when programming something bigger, advantage turns into disadvantage and the maintenance of the code is much more difficult. To prevent that I decided to use template at the global scope and Phenolphthalein library for individual components of the page.

I decided to use the PowerTemplate class as a template for the whole page. Reasons for choosing PowerTemplate were these (their order does not denote their importance).

- PowerTemplate is open-source and released under GPL.

[3]Please, note, that this shall not be meant as an offence against users using screen readers or text-browsers. It shall be rather meant as a description of users that use CSS and JavaScript capable graphics browser and do not have problems reading text from the screen

- It is very easy to use and offers everything I needed.
- It is a very small library and does not increase the load much.

I decided against Smarty – a very popular PHP templating engine – because it was unnecessarily complex for Lakmus and only fraction of its capabilities would be used.

The use of Phenolphthalein provided both markup separation and unified look. Originally, Phenolphthalein was library for creating forms, I added components for various listings, notebook-like panels later. That allowed to use Phenolphthalein in almost all parts of Lakmus where content generation was involved.

On the other hand, use of Phenolphthalein is not forced and programmer can decide whether to use it or whether direct HTML code would be better (in my opinion, it is *always* better to use Phenolphthalein because it makes the code more readable; often it is whether it is worth the effort to create a new Phenolphthalein component that would be used only once in the code).

3.5 Nice looking URLs

Debates whether nice looking URLs (or, better humanly-readable URLs) are important or worthless are evergreen between web developers. I belong to the group defending nice looking URLs because they are easier to remember and give user another hint on page categorization in the website[1].

For Lakmus I decided to use URL rewriting provided by Apache. As I use only a fraction of the possibilities of mod_rewrite it would be possible to move Lakmus to another web server without losing this functionality (for example, with Lighttpd rewriting would work as well).

As stated before, the parts of the URL (the virtual directories) describes the logical hierarchy and are used to determine which action shall be taken.

To simplify this, the `UrlDwarf` class exists that provides functions to handle URLs. This class is prepared by the main script and each part of Lakmus uses it to decide what page to display.

It has means to tell the programmer the name of the first (taking from the left) virtual directory and then shift the virtual pointer[4] behind that one (thus effectively hiding already processed part of the URL).

[4]This pointer always points to some directory separator and is used extensively for creating relative links

It is also used to create new links that are relative to the current one. By using already mentioned virtual pointer you can create links relative to any part of current URL. That it takes care of escaping of special characters is self-evident.

`UrlDwarf` is often used in functions like that (though, strings are usually replaced with constants as they are used also when creating links):

```
1  ...
2  switch ($this->url->getFirstSuffix()) {
3      case "edit":
4          $this->url->shift();
5          $this->doEdit();
6          break;
7      case "remove":
8          $this->url->shift();
9          $this->doRemove();
10         break;
11     case "":
12         $this->doIntro();
13         break;
14     ...
15 }
16 ...
```

The `getFirstSuffix` (line 2) tells the name of the first directory behind the virtual pointer; the `shift` (4 and 8) moves the pointer to the next directory. In the `doEdit` we would – for example – get the id of item being edited with call to `intval($this->url->getFirstSuffix())`.

Usage of `UrlDwarf` also gives a clean way to separate different actions and – in some kind – gives us the tools of event-driven programming.

3.6 Simple extensibility

The last highlight of Lakmus is its extensibility. Following chapter contains a fully commented process of writing a simple notice board application. Here, I would like to point out what makes the extensibility so simple.

When deciding how to add the applications, I recalled how it is done on application servers. You upload a package with certain format and you fill-in URL under which the deployed application would be accessible and several other

parameters. It is all very clean, intuitive and fool-proof as well. Because of this, I decided to do it similarly in Lakmus.

Fortunately, this was one of the last features I implemented – thus I knew what are the real demands and I did not fell into the trap of creating idealistic but non-implementable interface. I found out that the application always has the main implementation file (the one that extends the base class) and sometimes also has some kind of library files (such as data access wrappers) and quite often it needs to add it's own images and CSS definitions.

This lead to a creation of installer class from which each application derives its own and which takes care of the installation. To make the matter really simple, the base installer class offers only high-level methods such as `installLibraryFile` and thus to create the installer is very simple.

As mentioned above, each application may install a so-called library file. These files are stored in special directory where they are shared by all applications. This allows easy communication between different applications – only a simple file inclusion is needed to have access to "foreign" data. On the other hand, I dismissed completely any kind of dependency checking (i.e. to check that before installing application *xy* application *wz* has to be installed already) because it seems to me that Lakmus is not that kind of application that would really profit from having something like this. Definitely, the effort invested into implementation would be much bigger than the profit.

Also, except for two files that has to be present (the application identification and the installer file) the package does not need to keep a rigid structure.

As for the implementing itself, Lakmus provides a solid skeleton with functions to create the page content and to access core data (data about users, groups or labels). The set of available function is really rich and after you create wrappers to access data of your application, the implementation of the application is merely a chaining of various output components from the Phenolphthalein library.

And to simplify creation of a new application, I wrote several shell scripts – one is capable of creating the basic application skeleton, other then installs the application from command line (which is a noticeable speed-up contrary to manually uploading file to server through a web browser).

See appendix A for description of writing a simple noticeboard application.

Chapter 4

Existing applications

This chapter is about already implemented applications in Lakmus. Currently, there are four applications – these are application for sending e-mails, a simple forum, a so called scoreboard and tool for handling e-homework.

4.1 Mailer application

The Mailer application is used for sending e-mails to group members. As it is possible to label each group member, also e-mail sending use labels to tell which users will be recipients.

This application is very simple and small but it provides the expected functionality – it allows to send collective e-mails easily.

4.2 Phorum application

The Phorum application provides way for group communication between group members. Unlike "big" projects (such as phpBB), this application provides only a minimal set of features. However, as Lakmus is aimed at people who regularly met and electronic communication is merely a helper than main carrier, Phorum shall be sufficient. It allows any group member to start a new thread or to add a response. Additionally, group leader may mark topics as sticky and he can also mark topics as closed (to these, responses are prohibited). And, of course, it is possible to mark threads with labels and filter them using labels.

4.3 Scoreboard application

The Scoreboard application is a kind of spreadsheet targeted at keeping track of assigned points to individual group members. From some point of view, scoreboard is a traditional spreadsheet with a lot of features missing or eliminated. On the other hand, it has a feature that surpasses such tools.

As to to eliminated features: each user (group member) has a row "assigned" while individual categories (e.g. homework or tests) are in columns. Because each user must be treated equally, any function always apply the same way for the whole column and the function parameters are always taken from the same row. Also, number of functions is limited but from my personal experience (many lecturers used Google docs to keep track of our homework results) the only used function is sum.

The main goal of this application is the way columns are treated. In traditional spreadsheets, if you decide to have a sum column for all the homework, you say that a column is a sum of columns `A` to `F`. This approach has a very serious drawbacks: first, when we add a homework, we need to change the formula. The second problem is more fundamental – the letter (column names) does not denote in any way that they really hold a homework score.

With labels, both problems were solved using the same concept – labels. That is because each column could be tagged with labels and – which is to boom – the function does not operate on certain columns but on columns with specified label attached. Obviously, this approach may not be used for all functions but for the most typical – sum – it is a very clean solution that overpowers the classical one.

Obviously, there is place for improvements. An interesting extension might be to allow some kind of function parametrisation as currently function has only single parameter – the label describing columns to operate on. On the other hand, this application could be used successfully in the current state.

4.4 Homework application

The Homework application is aimed to be a replacement for sending homework solutions over e-mail – approach currently used by many lecturers.

The application provides the expected functionality – group leader announces tasks with descriptions set and users upload their solutions (set of files) to it – and adds a few (small, but useful) improvements.

A very handy extension is that user may upload more solutions than one. This feature was implemented because sometimes user wants to upload another solution (for example because he came up with an alternative algorithm) but does not want to overwrite his previous one (as he may not be sure about the new one). However, to prevent flooding the database, there is a hard-coded limit of how many solutions user may upload to each task.

As homework are usually evaluated and have points assigned, the homework application is able to cooperate with Scoreboard and transfer points to it. The solution scoring itself is distinctive, to say the least. First, each task has a number of maximum points assigned and each solution is graded in percentage. Next, each solution may be labeled and points may be added (or subtracted) using these labels as well.

Labels are used for adding bonus points and also for score degradation. The bonuses could be used to value an exceptional solution and these always hold an absolute value (i.e. they are not changed proportionally to the maximum points assigned). The score degradation could be used to decrease the maximum number of points for particular solution. This would be welcomed by lecturers who are very tough on deadlines and sometimes go as far that submitting solution 5 minutes after deadline means a 50% cut. Obviously, both ways of scoring could be used in reverse way also: bonus points could be negative (someone might prefer a subtract of fixed amount of points for late submit) and degradation could be higher than 1, thus giving a bonus proportional to the number of maximum points.

Among other features is possibility to mark solutions as public: these solutions then might be seen by all group members. To prevent embarrassment, the author name is undisclosed.

This application is one of the main highlights of Lakmus and even without other applications would make Lakmus an interesting tool to try.

Chapter 5

Security & safety

Here I would like to talk about decisions that I made during Lakmus implementation from the security point of view.

There are two possible angles to view the security of the application. First, it is the security towards users – how difficult is to access data without knowing log-in information and how difficult is to delete some data without warning? Secondly, it is the security when changing the code – how difficult is to corrupt the database when developing a new extension (new application).

5.1 Security for users

First, to access almost any page user has to be logged-in. There are two basic reasons for that: it is much easier to implement this approach than to implement that to implement content viewing with more levels of clearances. And also it hides data from random visitors because data stored in Lakmus are expected to be public (and relevant as well) only to small group of people – the community (students, developers...) that uses Lakmus.

User verification is done on pair username-password where both are stored in the same table on SQL server. The session itself is implemented via standard PHP sessions using cookies and thus the security issues are delegated to proper configuration of the web server[1] and PHP interpreter.

The password itself is stored as SHA1 sum of mangled password. The password mangling is done as follows. Password is split into halves and random (though fixed) string is inserted between them. This new password is appended

[1] That could also include running Lakmus under secured HTTP protocol only.

to the username and this string is then hashed.

Having the hash base computed like this will make creation of any rainbow table much more difficult [5, comment by Kevin]. The first salt is different for each user so it is not possible to "pre-hash" some part of the password. Next, the effort for getting the second salt is equivalent to web server breach (as this salt is stored in a configuration file). And again, inserting the second salt inside the password effectively disables any hash pre-processing.

5.2 Security when developing

Contrary to security for users, internal security for developers is minimal and is set more on recommendations than on restrictive checks. In other words, approach that for well-educated people sign "No entrance" is enough was taken.

I decided for this approach because, so far, this project is a "one-man show" and any extra barriers would only limit the development. On the other hand, no part of the Lakmus abuses this and if a need arises to introduce a new security measures, the actual changes to the code would have a very low (if some) overhead.

As to make it clear what I meant by this "security" it means that all parts of the application has read/write access to all data available. Obviously, this approach is a necessity for the kernel. With applications, it is different. Application is a per-group entity and thus the common sense is to allow it to read information about the group (e.g. members' names) and to allow a read/write access to the labels. And, of course, allow it to manage its own data.

Nevertheless, this strict approach would effectively forbade any simple inter-application communication (as there would be need to create some kind of messenger – in the kernel – that would pass data from one application to another), not talking about some special-purpose applications (whatever that could mean).

In my opinion, the "well-educated knows" approach is currently the best solution because it allowed me to programme the applications very quickly yet it does not prevent any further changes that would restrict what applications can and can't do.

5.3 Further development

Although this section will not be on security only, it belongs in this part of the paper as the most discussed part will concert security enhancements.

As stated before, Lakmus provides a rich framework that should help create new applications easily and quickly. As the only limitation what applications could be created is the fantasy of potential users, it would be useless to discuss what else Lakmus may offer. There is one application, though – application for some kind of automated on-line examination might be very useful (however, leaving out speculations whether examination that could be automatically corrected really examine somebody).

Possible enhancements in other aspects may include more restricted data access control for the applications. First, it may be possible to create for applications different SQL users that would not have write access to core data. However, applications must have a write access to labels and all labels (no matter what are they attached to) are in single table and simple yes-no control on table would not prevent malicious application to destroy labels attached to system entities (e.g. to groups, applications etc.).

Then there comes the problem of inter-application communication. If the access would be only to data created by the application, the homework-scoreboard application duo would not be able to cooperate (unless some kind of messenger – as stated above – would be created).

Another approach to this might be to allow administrators to mark applications as safe – these then could access all the data while the unsafe could access only their own.

All these possibilities are technically possible and they might improve Lakmus. However, the question is whether the effort invested is worth gained advantages.

For me, the answer is no. I have three reasons for this (and omitting it would mean more code writing for me). First, Lakmus is not a primary (crackers') target as the data are relevant to only a very small group of people, moreover; they are relevant only for a small amount of time (e.g. a semester). Next, Lakmus's security is much delegated to the used web and database server and it is better to harden these servers than Lakmus itself (that does not mean that I think Lakmus could be left without improvements but in my opinion, it is better investment to use time to install necessary patches to the web server, configure SSL etc. than to create a sense of false security by patching a software that has source-code available). Finally, breach is always possible and you always need to check installed applications[2].

[2]If they are from credible sources, you do not need any extra counter-measures and when not, you need to check them anyway and the counter-measures would be redundant after the check.

Anyway, if a need for such changes would arise, the layered structure would make such actions quite simple. For example, to add checks that current user (i.e. the one that is logged-in to the page that is executed) has the right to do something, simple call to method `LakmusAuthorizator` would be needed to add to the beginning of the appropriate function (e.g. in `LakmusLabelMate` or in `LakmusCoreDataHelper`). If the credentials won't be sufficient, an exception might be thrown and the exception would be then caught in the top-most code and error would be displayed. As could be seen, this would effect only the middle layer. Other option (though, due to diversity among SQL `SELECT` queries this would be more difficult to implement[3]) might be to check on the lowest layer that user has rights to execute each query.

[3]Because either a new way to create the `SELECT` queries must be used or the queries needed to be syntactically parsed

Appendix A

Lakmus extensibility

As stated before, Lakmus is also targeted at easy extensibility and it's policy is to encourage users to write their own applications[1] to extend the functionality.

Guide on extending Lakmus is included in User's manual, here I would like to show how the principles mentioned previously lead to an easy way of extending the Lakmus without need of an excessive studying of used libraries. Also, I will skip the part on how to technically add our application to Lakmus.

I would demonstrate the simplicity on writing a simple notice board. As the focus shall be more on the code than the functionality, the written application would have only these features:

- notice board would be used by group leader (administrator) to inform users about last-minute changes
- each notice would consist of a title, body and time of creation (or last edit)
- only administrator can create and edit notices
- the first page of the application would display n newest notices (having n as a constant hard-coded in the application) and a link to older ones

A.1 Data storage

Because our notice board is very simple, we would suffice with a single table – `lkma_noticeboard` – with these columns:

[1] By the word *application* in this context is meant a group application (plug-in).

id notice unique id, primary key (integer)

title title of the notice (varchar)

details notice text (varchar)

updated time of last update (here we can make the best of by using MySQL native type for timestamps)

author id of user who entered the notice – actually not necessarily needed as only admins can insert notices but won't make any harm; foreign key to `lkm_user` (integer)

gid id of group the notice belongs to, foreign key to `lkm_group` (integer)

As stated in chapter 3, Lakmus uses three-layered structure for data operations. As this is a good practise, we would create a wrapper class around notice board data called `NoticeBoardDataWrapper`. There is nothing magic in it as you can see on the listing and in most functions it only wraps calls to the SQL server (plus some extra checks).

```
1  class NoticeBoardDataWrapper {

3  private $sql; // SqlConnection
4  private $gid; // int

6  public function __construct(&$sql, $gid) {
7      assert(is_int($gid));8
8      $this->sql =& $sql;
9      $this->gid = $gid;
10 }

12 public function getNewest($count) {
13     assert(is_int($count));
14     return $this->sql->select("title, details, author "
15         . "FROM lkma_noticeboard "
16         . "WHERE gid=? LIMIT ?", array($gid, $count));
17 }

19 public function getAll() {
20     return $this->sql->select("title, details, author "
```

```
21          . "FROM␣lkma_noticeboard␣"
22          . "WHERE␣gid=?", $gid);
23  }

25  public function addNotice($info) {
26      assert(is_struct($info,
27          array("title", "details", "gid", "author"), TRUE));
28      $this->sql->insert("lkma_noticeboard", $info);
29  }

31  public function updateNotice($id, $info) {
32      assert(is_struct($info,
33          array("title", "details", "gid", "author"), TRUE));
34      assert(is_int($id));
35      $this->sql->update("lkma_noticeboard",
36          $info,
37          array("id" => $id));
38  }


41  }
```

A.2 Start up

Each application in Lakmus is a class derived from `LakmusApplicationBase`. A glance at its documentation will reveal that there are 5 public methods, public constructor, quite a lot of protected methods and a bunch of protected attributes (mostly other classes). In following paragraphs, the word *base* would refer to that class. Additionally, when taking about the class we derive (i.e. the one that implements our notice board application), it would be referred as *our* class.

We will start with a closer look at initialization. The constructor of the base class takes only one parameter – application name. That shall be a human readable name of the application. When everything goes fine, this name would be never displayed as it is displayed in error messages (and these would be displayed only if you forget to catch some exception which would thus propagate to `Lakmus GroupActions` which owns all group applications). Our constructor does not have any parameters and its body shall consist only of a call to a parent constructor.

If we need to make more complex initializations, they have to wait to the `onInit` function. That method is called after constructing the object. The reason for not doing that in the constructor is that data wrappers are yet not prepared there. Originally, all (at that time only two) data wrappers were passed as parameters to the constructor. However, when `LakmusLabelMate` was separated from `LakmusCoreDataHelper`, it meant that all constructors of all applications had to be rewritten – to avoid this happening again, these data wrappers are passed to our class in a separate function call (that we do not need to worry about).

The only preparation we need to take is to initialize our `NoticeBoardDataWrapper` class.

A.3 Available helpers and wrappers

When `onInit` is called, following helpers are available:

`auth` authorization manager – holds information about current user (`LakmusAuthorizator`)

`data` wrapper to core data – e.g. to information about users and groups (`LakmusCoreDataHelper`)

`labels` wrapper for accessing labels (`LakmusLabelMate`)

`sql` wrapper around initialized connection to an SQL server (`SqlConnection`)

`url` helper for parsing URLs – used extensively for determining which action shall be taken (`UrlDwarf`)

`ginfo` structure with information about current group (you shall threat this attribute as read-only)

A.4 Handling actions

The heart of the application is the `doAction` function. This function is called after necessary preparations and is expected to prepare content of the page and – if needed – update data.

In our application, we would in this function determine what user wants to do and call other function to handle it. To determine the action requested we

would look at the URL of current page. Using the `getSingleSuffix` function of the `url` attribute, we would get the first virtual directory in the URL that was passed.

To clarify it, suppose browser displays page *http://your-host/lakmus/group/ /100/app/noticeboard/admin/edit/2*. The kernel of Lakmus parses and processes[2] the part from *group* onwards. The *100* would be interpreted as group id, the *app* starts application part and *noticeboard* would be interpreted as application to be started. Everything else (in our example *admin/edit/2*) is passed to the application itself inside the `UrlDwarf` (it could be imagined that there is a pointer that indicates which parts in the URL shall be processed). Back to the `getSingleSuffix` – it would return the first directory: in our example it would be *admin*.

In our application, we would have only four actions – the "intro" page with newest notices, the archive page for older posts and the administration pages – the edit page and the "add new notice" page.

We would switch on them like this (also notice we added them as constants):

```
1  public function doAction() {
2      switch ($this->url->getSingleSuffix()) {
3          case self::LINK_ADDNOTICE:
4              $this->url->shift();
5              $this->displayAddNoticePage();
6              break;
7          case self::LINK_EDIT:
8              $this->url->shift();
9              $this->displayEditNoticePage();
10             break;
11         case self::LINK_ARCHIVE:
12             $this->url->shift();
13             $this->displayArchivePage();
14             break;
15         case "":
16             // fall-through to default
17         default:
18             $this->displayIntroPage();
19             break;
20     }
```

[2]because Lakmus uses URL rewriting a lot, URL has to parsed inside Lakmus as all parts of the URL are virtual directories that really does not exist

```
21  }
```

So far, there is nothing incomprehensible in this. The `shift` on the URL wrapper causes the internal pointer to be shifted to next directory[3]. For now, we would silently ignore a unknown action and simply display the "intro" page.

A.4.1 The "intro" page

On this page we want to display the newest notices, link to the archive and also links for editing (of course, only when user is administrator of the group).

Let's create the link for adding a new notice first. By the character of the link, the best choice is to use one of Phenolphthalein components – the `Phnl ActionList`. This component creates a list of links and in modern browsers the unordered list is converted into horizontal boxes with small icons (this component is used, for example, on your home page in Lakmus). Our function would look like this:

```
22  protected function displayIntroPage() {
23      $isGroupAdmin = $this->auth->isGroupAdmin(
            $this->ginfo['id']);

25      if ($isGroupAdmin) {
26          $toplinks = new PhnlActionList();
27          $toplinks->add($this->url->get("/" .
           self::LINK_ADDNOTICE),
               "Add␣notice", "admin");
28          $this->setPageContent($toplinks->getAsHtml());
29      }

31      // more code here to come
32  }
```

Parameters to the `add` (line 27) are pretty obvious – it is the URL of the page link points to, the link name and CSS class that is used to determine the icon. The predefined "admin" class will display a tool icon.

The `setPageContent` will set the content of the page we are creating. Later on, we will use `appendPageContent` to append other content.

[3]Meaning that when the URL ends with *..../admin/edit/2*, after the shift, `getSingle Suffix` would return *edit*.

Let's display the notices. One of the advantages in Lakmus is that you can easily mix predefined components with your "hand-made" HTML content. Although, best practice is avoid such thing, here we will do it when creating the list of notices (and as the archive listing would not differ in its output much, separate function – returning generated HTML code – was created).

```
33  protected function createNoticeListing(
        $notices, $displayEditLinks) {
34      if (count($notices) > 0) {
35          $noticeListing = "<dl>";
36          foreach ($notices as $notice) {
37              $noticeListing .= sprintf(
                    "<dt>%s</dt>\n<dd>%s</dd>\n",
38                  htmlspecialchars($notice['title']),
39                  htmlspecialchars($notice['details']));
40              if ($displayEditLinks) {
41                  $noticeListing .= sprintf(
                        '<dd><a␣href="%s">%s</a></dd>',
42                      $this->url->get("/" . self::LINK_EDIT
                            . "/" . $notice['id']),
43                      "edit");
44              }
45          }
46          $noticeListing .= "</dl>";
47          return $noticeListing;
48      } else {
49          return LkmPhnl::emptyResults(
                "There␣are␣no␣notices."));
50      }
51  }


54  protected function displayIntroPage() {
55      $isGroupAdmin = $this->auth->isGroupAdmin(
            $this->ginfo['id']);

57      if ($isGroupAdmin) {
58          $toplinks = new PhnlActionList();
59          $toplinks->add($this->url->get(
```

```
            "/" . self::LINK_ADDNOTICE),
            "Add␣notice", "admin");
60          $this->setPageContent($toplinks->getAsHtml());
61      }

63      $notices = $this->notices->getNewest(
            self::INTRO_PAGE_NOTICE_COUNT);
64      $this->appendPageContent(
            $this->createNoticeListing($notices, $isGroupAdmin));

66      if (count($notices) < $this->notices->getCount()) {
67          $this->appendPageContent(sprintf(
                '<p><a␣href="%s">%s</a></p>',
68              $this->url->get("/" . self::LINK_ARCHIVE),
69              "See␣older␣posts"));
70      }
71  }
```

Again, this code does not bring anything extra – the intro page is built. However, several things needed to be pointed out.

- Output is escaped with `htmlspecialchars` as seen on lines 38 and 39 because all data wrappers return "raw" strings. This approach is better than escaping data inside the wrappers because the wrappers themselves do not know which kind of escaping shall be done (some applications have – for example – output into LaTeX which uses completely different kind of escaping than HTML).

- The `emptyResults` (see line 49) is a static function of `LkmPhnl` class intended to display notices about empty result set (in text-browsers, no formatting is handled; in modern ones, different color indicates its purpose).

- For creating HTML content, the `sprintf` is always used because it allows separation of markup from strings which could much simplify later localization (as we would merely enclose the strings in `_(...)`.

A.4.2 The "Archive" page

This function needs no explanation and is here only for completeness.

```
72  protected function displayArchivePage() {
73      $isGroupAdmin = $this->auth->isGroupAdmin(
            $this->ginfo['id']);

75      // edit links are expected from the root
76      $this->url->unshift();

78      $toplinks = new PhnlActionList();
79      $toplinks->add($this->url->get("/"),
            "Back␣to␣newest␣ones␣only", "back"))
80      if ($isGroupAdmin) {
81          $toplinks->add($this->url->get(
                "/" . self::LINK_ADDNOTICE),
                "Add␣notice", "admin");
82      }
83      $this->setPageContent($toplinks->getAsHtml());

85      $notices = $this->notices->getAll();
86      $this->appendPageContent(
            $this->createNoticeListing($notices, $isGroupAdmin));
87  }
```

A.4.3 Adding and editing notices

Although adding and editing a notice are presented as two different actions, they are pretty much the same. The would use the same HTML form – the only difference would be that "edit notice"s form would have already filled in the field values. Because of this, a single function that handles displaying and running the form would be used for both actions.

With such function, the `displayAddNoticePage` would look like this:

```
88  protected function displayAddNoticePage() {
89      $this->checkAdministratorship();

91      $this->displayGeneralizedNoticeEditingPage(array(
92              "id" => FALSE, // indicator of notice adding
93              "title" => "",
94              "details" => ""),
95          $this->url->get("/"),
```

```
96          "Add␣a␣notice", "Add␣this␣notice",
97          "Notice␣%s␣was␣added.");
98  }
```

The meaning of parameters of `displayGeneralizedNoticeEditingPage` is displayed later. The `checkAdministratorship` (line 89) needs a bit of explanation. It checks that current user is administrator of current group and when not an exception is thrown. Because we do not catch it anywhere, it would propagate up to the owner of our class (the `LakmusGroupActions`) where it would be intercepted and an error message would be displayed. As we do not create any special layout that might be broken by this, we could simplify the job by using this function. If such behaviour would not be acceptable you can always catch the exception somewhere or instead insert a simple "if" check with `isGroupAdmin`.

The `displayEditNoticePage` function is a bit longer because we have to check that the provided id is valid.

```
99  protected function displayEditNoticePage() {
100     $this->checkAdministratorship();

102     $noticeId = intval($this->url->getSingleSuffix());

104     $noticeDetails = $this->notices->getNoticeDetails(
            $noticeId);
105     if (($noticeDetails !== FALSE)
            && ($noticeDetails['gid'] == $this->ginfo['id'])) {
106         $this->displayGeneralizedNoticeEditingPage(
                $noticeDetails,
107             $this->url->get("/" . $noticeDetails['id']),
108             sprintf(
                    "Edit␣%s␣notice",
                    LkmPhnl::eemp($noticeDetails['title'])),

109                 "Store␣the␣changes",
110                 "Notice␣%s␣was␣updated.");
111     } else {
112         $this->url->unshift();
113         $this->appendPageContent(Phnl::getErrorBox(
114             sprintf(_("You␣wanted␣to␣edit␣a␣notice␣
                    which could not be found. Please,
```

```
                return to the <a href=\"%s\">list
                of them</a>."


117 ),
118             $this->url->get("/"))));
119     }
120 }
```

The only thing worth noticing is the usage of `getErrorBox` (113) that returns HTML code with formatted error message. The `eemp` on line 108 is used to emphasise and escape a string[4].

Now it is the time to see the implementation of `displayGeneralizedNotice EditingPage`.

```
121 protected function displayGeneralizedNoticeEditingPage(
        $noticeDetails, $formAction, $formCaption,
        $formSubmitCaption, $newsflashMessage) {
122     assert(is_array($noticeDetails));
123     ... // more asserts

125     $form = new PhnlForm2($formAction, "post");
126     $form->setCaption($formCaption);
127     $form->setSubmitCaption($formSubmitCaption);
128     $form->setSentFlag("sent", "1");

130     $noticeTitle = new PhnlTextInput("title",
            "Notice␣title", ...);
131     $noticeTitle->setHint("Insert␣an␣eye-catching␣title");

133     $noticeDetails = new PhnlTextInput("details",
            "Notice␣details", ...);
134     $noticeDetails->setMultiline();
135     $noticeDetails->setHint("Write␣here␣extra␣information");

137     $form->addField($noticeTitle);
138     $form->addField($noticeDetails);
```

[4]Its body consists merely of call to `htmlspecialchars` and adding the `<em>` tag around returned string.

```
140     if ($form->wasSent()) {
141         $form->readData();
142         $form->validate();
143         if ($form->isValid()) {
144             $data = $form->getData();
145             // add current group id and UID of author
146             $data['gid'] = $this->ginfo['id'];
147             $data['author'] = $this->auth->getUid();
148             if ($noticeDetails['id'] === FALSE) {
149                 $this->notices->addNotice($data);
150             } else {
151                 $this->notices->updateNotice(
                        $noticeDetails['id'], $data);
152             }
153             Phnl::sendNewsflash(sprintf($newsflashMessage,
154                 LkmPhnl::eemp($data['title'])));
155             $this->url->redirectRoot();

157         }
158     } else {
159         $form->setData($noticeDetails);
160     }

162     $this->appendPageContent($form->getAsHtml());
163 }
```

As could be seen on line 122 we check for correct type of the parameters (see chapter 3).

Starting on line 128 up to 128 we create a Phenolphthalein form component. The "sent flag" is used to determine whether form was already sent.

Then we create the text input field (130) and set a floating help hint.

The input field for notice details (133) is set as multi-line (thus producing a TEXTAREA instead of INPUT).

These fields are then added to the form.

On line 140 we check that form was already sent and when it was, we read the data from the `_POST` array and then validate them (e.g. check that required fields are not blank or that match given regular expression).

If the data are valid, we get them (144) as an associative array. To this

array we add information about current group and author and then – depending whether the id was known – either update or add the notice.

When the data are stored – to prevent unintentional re-do of the action – page is redirected to the "intro" one (line 155).

However, we want to inform user that data were stored. To do that we send so called "newsflash" that is stored as a session attribute and is displayed on next page. All that we need to do is to call `Phnl::sendNewsflash` (153) with appropriate message (that also explains that `%s` in the parameters.

A.4.4 Conclusion

That is all. With less than 200 lines we were able to implement a very simple (yet functional) notice board. As you may noticed, the key feature of Lakmus, the labels, were not covered. There are two reasons for that: first, our notice board is to simple to make them useful and they require more code. However, their usage when developing is described in User's manual.

Appendix B

Attachments

- Printed user's manual for Lakmus
- CD with
  - "Gzipped" TAR archives with Lakmus sources for installation (referenced from the manual)
  - Four Lakmus applications
  - Generated API documentation
  - Manual sources and generated PDF

Bibliography

[1] Tim Berners-Lee *Cool URIs don't change*
http://www.w3.org/Provider/Style/URI

[2] *PHP Framework Benchmarks*
http://www.sellersrank.com/web-frameworks-benchmarking-results/

[3] Ekerete, AVNet Labs *PHP framework comparison benchmarks*
http://avnetlabs.com/php/php-framework-comparison-benchmarks

[4] The PHP Group *PHP Usage Stats*
http://www.php.net/usage.php

[5] The PHP Group *PHP: hash (user's comments)*
http://en.php.net/manual/en/function.hash.php

[6] Dan Cederholm *Web Standards Solutions: The Markup And Style Handbook*,
Apress L.P. 2004

[7] Peter Nederlof *Whatever:hover*
http://www.xs4all.nl/~peterned/csshover.html